|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SFT 221 | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Workshop 1 | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Name: | Ankush Gauro | | | | | | | |
| ID: | 108593237 | | | | | | | |
| Email: | agauro@myseneca.ca | | | | | | | |
| Section: | ZFF | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Authenticity Declaration: I declare this submission is the result of my own work and has not been shared with any other student or 3rd party content provider. This submitted piece of work is entirely of my own creation. | | | | | | | | |

**Testing code**:

int startsWith(const char s[], const char prefix[])

{

char buf[25];

int i;

int sz = strlen(prefix);

for (i = 0; i < sz; i++)

{

buf[i] = s[i];

}

buf[sz] = '\0';

return 0 == strcmp(buf, prefix);

}

int endsWith(const char s[], const char suffix[])

{

int sz = strlen(suffix);

int slen = strlen(s);

return 0 == strcmp(s + slen - sz, suffix);

}

**Description:** Here, I was just running the whole program and checking if it was working correctly i.e. if the functions startsWith and endsWith deliver the preferred output for the given default inputs in the workshop 1 pdf file (weather “upended” starts with “up” and ends with “ed”).

**Result:** It passes the test cases and the result was,

upended does start with up

upended does end with ed.

No bugs were identified by these tests.

**Identified bugs:**

**Bug : Buffer Overflow and ensuring sizes of ‘s’ and ‘buf’**

After inspecting the code, I found out a bug with buffer overflow in the startsWith function. This function doesn’t consider the size of the string ‘s’ and doesn’t check if ‘s’ is longer than the allocated memory ‘buf’. This could lead to buffer overflow and undefined behaviours.

**Fixes:**

|  |  |
| --- | --- |
| Before | After |
| int startsWith(const char s[], const char prefix[])  {  char buf[25];  int i;  int sz = strlen(prefix);  for (i = 0; i < sz; i++)  {  buf[i] = s[i];  }  buf[sz] = '\0';  return 0 == strcmp(buf, prefix);  } | int startsWith(const char s[], const char prefix[]) {  int sz = strlen(prefix);  if (strlen(s) < sz) return 0;  return strncmp(s, prefix, sz) == 0;  } |

Here, I fixed the code and used ‘strncmp’ to directly compare the ‘prefix’ with the first ‘sz’ characters of ‘s’. This fix counters the bug of buffer overflow and this fix also includes an if check to ensure that ‘s’ is at least as long as ‘prefix’ before comparing and it will return 0 if it is not.

**Bug : Doesn’t handle empty strings**

One more bug that I found in this code is that it doesn’t handle empty strings which could lead to unexpected behaviour in the program.

**Fixes:**

|  |  |
| --- | --- |
| Before | After |
| int startsWith(const char s[], const char prefix[])  {char buf[25];  int i;  int sz = strlen(prefix);  for (i = 0; i < sz; i++)  {  buf[i] = s[i];  }  buf[sz] = '\0';  return 0 == strcmp(buf, prefix);  }  int endsWith(const char s[], const char suffix[])  {  int sz = strlen(suffix);  int slen = strlen(s);  return 0 == strcmp(s + slen - sz, suffix);} | int startsWith(const char s[], const char prefix[])  {  if (strlen(prefix) == 0) {  return 0; }  char buf[25];  int i;  int sz = strlen(prefix);  for (i = 0; i < sz; i++)  {  buf[i] = s[i];  }  buf[sz] = '\0';  return 0 == strcmp(buf, prefix);  }  int endsWith(const char s[], const char suffix[])  {  if (strlen(suffix) == 0) {  return 0; }  int sz = strlen(suffix);  int slen = strlen(s);  return 0 == strcmp(s + slen - sz, suffix);  } |

Here, I added a conditional if block to ensure that the strings are not empty.

**Bug: Array initialization**

The array initialization for s1[], prefix[] and suffix[] in the main function is redundant and can be simplified.

|  |  |
| --- | --- |
| Before | After |
| char s1[] = { "upended" };  char prefix[] = { "up" };  char suffix[] = { "ed" }; | char s1[] = "upended";  char prefix[] = "up";  char suffix[] = "ed"; |

Here, I just removed the unnecessary curly braces and initialized s1, prefix and suffix directly. I know it’s not a critical bug but it improves the code readability.

**Reflection:**

In this case, testing initially identified no bugs, but inspection revealed the buffer overrun issue in the startsWith function. I think inspection worked better for me than testing in this case because this was a relatively smaller program and hence the inspection of the program was much easier. However if this were a big program, testing it with various cases would have helped a lot in finding potential bugs.

To enhance testing, including edge cases where string lengths are at the limit or exceed buffer sizes could be beneficial. Also, using test cases with empty strings could help in identifying potential issues that might not be apparent initially.

Finding bugs in this assignment wasn’t relatively difficult. This is because is was a pretty small program and all the bugs were quite straightforward. In this case, inspecting the code line by line, understanding the code behaviour, running tests and reviewing the outputs helped me a lot in finding the bugs in this program. However, in a relatively larger program bugs might not be immediately obvious without deeper code inspection combined with testing it with various testing scenarios.